A Gentle Adventure Mechanising Message Passing Concurrency Systems

An Experience/Walkthrough Report

David Castro-Perez, Lorenzo Gheri, Francisco Ferreira, Martin Vassor, and Nobuko Yoshida

DisCoTec22

Imperial College London

University of Kent

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Roadmap

1. Binders and Linearity

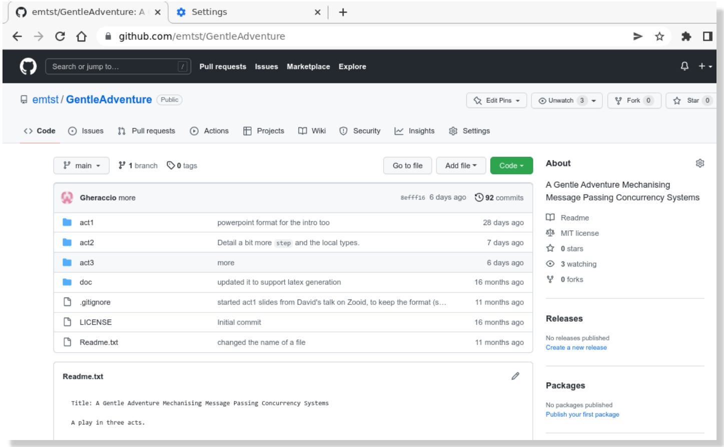
Act I: Binary Session Types

2. Multiparty Processes and Coinduction

Act II: Mechanising Multiparty Processes

Act III: Mechanising Multiparty Session Types

SmolEMTST: Tutorial Repository



Gheraccio more 8efff16 6 days ago 92 commits

act1	powerpoint format for the intro too	28 days ago
act2	Detail a bit more `step` and the local types.	7 days ago
act3	more	6 days ago
doc	updated it to support latex generation	16 months ago
.gitignore	started act1 slides from David's talk on Zooid, to keep the format (s...	11 months ago
LICENSE	Initial commit	16 months ago
Readme.txt	changed the name of a file	11 months ago

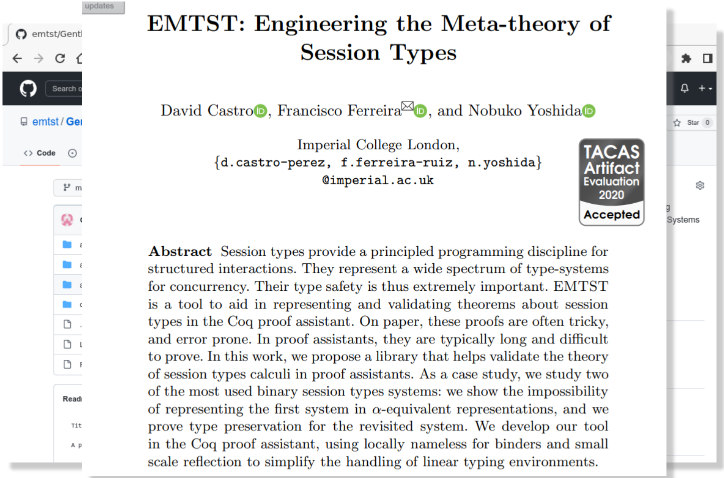
A Gentle Adventure Mechanising Message Passing Concurrency Systems

Readme.txt

```
Title: A Gentle Adventure Mechanising Message Passing Concurrency Systems

A play in three acts.
```

SmolEMTST: Tutorial Repository



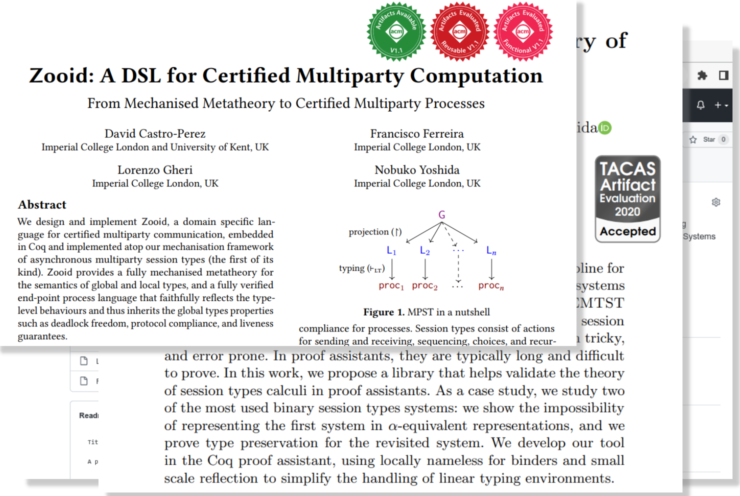
EMTST: Engineering the Meta-theory of Session Types

David Castro, Francisco Ferreira, and Nobuko Yoshida

Imperial College London,
{d.castro-perez, f.ferreira-ruiz, n.yoshida}
@imperial.ac.uk

Abstract Session types provide a principled programming discipline for structured interactions. They represent a wide spectrum of type-systems for concurrency. Their type safety is thus extremely important. EMTST is a tool to aid in representing and validating theorems about session types in the Coq proof assistant. On paper, these proofs are often tricky, and error prone. In proof assistants, they are typically long and difficult to prove. In this work, we propose a library that helps validate the theory of session types calculi in proof assistants. As a case study, we study two of the most used binary session types systems: we show the impossibility of representing the first system in α-equivalent representations, and we prove type preservation for the revisited system. We develop our tool in the Coq proof assistant, using locally nameless for binders and small scale reflection to simplify the handling of linear typing environments.

SmolEMTST: Tutorial Repository



Zooid: A DSL for Certified Multiparty Computation

From Mechanised Metatheory to Certified Multiparty Processes

David Castro-Perez
Imperial College London and University of Kent, UK

Francisco Ferreira
Imperial College London, UK

Lorenzo Gheri
Imperial College London, UK

Nobuko Yoshida
Imperial College London, UK

Abstract

We design and implement Zooid, a domain specific language for certified multiparty communication, embedded in Coq and implemented atop our mechanisation framework of asynchronous multiparty session types (the first of its kind). Zooid provides a fully mechanised metatheory for the semantics of global and local types, and a fully verified end-point process language that faithfully reflects the type-level behaviours and thus inherits the global types properties such as deadlock freedom, protocol compliance, and liveness guarantees.

Figure 1. MPST in a nutshell

compliance for processes. Session types consist of actions for sending and receiving, sequencing, choices, and recur-

ry of

ida

TACAS Artifact Evaluation 2020 Accepted

pline for
systems
EMTST
session
tricky,

and error prone. In proof assistants, they are typically long and difficult to prove. In this work, we propose a library that helps validate the theory of session types calculi in proof assistants. As a case study, we study two of the most used binary session types systems: we show the impossibility of representing the first system in α-equivalent representations, and we prove type preservation for the revisited system. We develop our tool in the Coq proof assistant, using locally nameless for binders and small scale reflection to simplify the handling of linear typing environments.

Mechanising the **Honda, Vasconcelos and Kubo's binary session type system**

Honda, K., Vasconcelos, V. T., and Kubo, M. (1998). Language primitives and type discipline for structured communication-based programming. In Hankin, C., editor, Programming Languages and Systems, pages 122–138, Berlin, Heidelberg. Springer Berlin Heidelberg

Processes: Key Features

$$\begin{array}{rcll}
P, Q, R & ::= & & \\
& \mid & k\,![e];\ P & \text{data sending} \\
& \mid & k\,?(x).P & \text{data receiving} \\
& \mid & \mathtt{throw}\ k\,[k'];P & \text{channel sending} \\
& \mid & \mathtt{catch}\ k\,(k').P & \text{channel receive} \\
& \mid & P \mid Q & \text{parallel composition} \\
& \mid & \nu_c\,(k).P & \text{channel hiding} \\
& \mid & \ldots &
\end{array}$$

Processes: Key Features

$$
\begin{array}{rcll}
P, Q, R & ::= & & \\
& \mid & k\,!\,[e];\ P & \text{data sending} \\
& \mid & k\,?\,(x).P & \text{data receiving} \\
& \mid & \mathtt{throw}\ k\,[k'];\,P & \text{channel sending} \\
& \mid & \mathtt{catch}\ k\,(k').P & \text{channel receive} \\
& \mid & P \mid Q & \text{parallel composition} \\
& \mid & \nu_c\,(k).P & \text{channel hiding} \\
& \mid & \ldots &
\end{array}
$$

Binder Mechanisation: DeBruijn Indices

Terms $M, N ::= n \mid M\ N \mid \lambda.\, M$

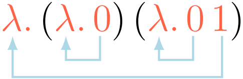
Binder Mechanisation: DeBruijn Indices

Natural number

Terms $M, N ::= \overset{\text{Natural number}}{n} \mid M\ N \mid \lambda .\, M$

Binder Mechanisation: DeBruijn Indices

Terms $\qquad M, N ::= n \mid M\,N \mid \lambda.\,M$



$$\lambda.\,(\lambda.\,0)\,(\lambda.\,0\,1)$$

Binder Mechanisation: Locally Nameless

Natural number

Name that represents a free variable

Terms: $M, N ::= n \mid x \mid M\,N \mid \lambda.\,M$

Binder Mechanisation: Locally Nameless

Terms: $M, N ::= n \mid x \mid M\ N \mid \lambda.\ M$

$M^{x} \equiv \{0 \to x\}M$	Open a term
$\backslash^{x} M \equiv \{0 \leftarrow x\}M$	Close a term
$\mathsf{lc}(M)$	Locally closed term

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \qquad \frac{\forall x \notin L \qquad \Gamma, x : S \vdash M^{x} : T}{\Gamma \vdash \lambda.\ M : S \to T}$$

$$\frac{\Gamma \vdash M : S \to T \qquad \Gamma \vdash N : S}{\Gamma \vdash M\ N : T}$$

Process Mechanisation

```
Inductive name : Set :=
  | fnm : atom → name
  | bnm : nat → name
.

Definition channel := name.

Inductive proc : Set :=
| send : channel → exp → proc → proc
| receive : channel → proc → proc

| throw : channel → channel → proc → proc
| catch : channel → proc → proc

| nu_ch : proc → proc (* hides a channel *)
...
```

```
Inductive lc : proc → Prop :=
| lc_send : forall k e P,
    lc_nm k →
    lc_exp e →
    lc P →
    lc (send k e P)

| lc_receive : forall (L : seq atom) k P,
    lc_nm k →
    (forall x, x \notin L → lc (open P x)) →
    lc (receive k P)


| lc_throw : forall k k' P,
    lc_nm k → lc_nm k' →
    lc P →
    lc (throw k k' P)

| lc_catch : forall (L : seq atom) k P,
    lc_nm k →
    (forall x, x \notin L → lc (open P x)) →
    lc (catch k P)
...
```

Semantics (Excerpt as in Paper)

R-PASS	$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,(k').Q \to P \mid Q$
R-COM	$k\,![e];\ P \mid k\,?(x).Q \to P \mid \{x \to e\}Q$
R-CONG	$P \equiv P'\text{and } P' \to Q'\text{and } Q' \equiv Q \Rightarrow P \to Q$
R-SCOP	$P \to Q \Rightarrow \nu_c\,(k).P \to \nu_c\,(k).Q$
R-PAR	$P \to P' \Rightarrow P \mid Q \to P' \mid Q$

Mechanising the Semantics

Using Locally Nameless, α-equivalent terms are **syntactically equal**

How do we mechanise R-PASS?

$$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,(k').Q \to P \mid Q$$

Mechanising the Semantics

Using Locally Nameless, α-equivalent terms are **syntactically equal**

How do we mechanise R-PASS?

$$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,(k').Q \to P \mid Q$$

A naive attempt:

$$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,().Q \to P \mid Q^{k'}$$

Mechanising the Semantics

Using Locally Nameless, α-equivalent terms are **syntactically equal**

How do we mechanise R-PASS?

$$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,(k').Q \to P \mid Q$$

A **WRONG** attempt:

$$\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,().Q \to P \mid Q^{k'}$$

Why is our mechanisation of r-pass wrong?

$$\begin{array}{llcl} \text{Type} & \alpha, \beta & ::= & ![S];\alpha \mid ?[S];\alpha \mid \mathbf{end} \mid \bot \\ \text{Typing} & \Delta & ::= & \cdot \mid \Delta, k : \alpha \end{array}$$

Subject Reduction: if $\Gamma \vdash P \triangleright \Delta$ with Δ balanced, and $P \to Q$, then there exists Δ' s.t. $\Gamma \vdash Q \triangleright \Delta'$, with Δ' balanced.

Why is our mechanisation of r-pass wrong?

$$\begin{array}{lrcl} \text{Type} & \alpha, \beta & ::= & ![S]; \alpha \mid ?[S]; \alpha \mid \mathbf{end} \mid \bot \\ \text{Typing} & \Delta & ::= & \cdot \mid \Delta, k : \alpha \end{array}$$

Subject Reduction: if $\Gamma \vdash P \triangleright \Delta$ with Δ balanced, and $P \to Q$, then there exists Δ' s.t. $\Gamma \vdash Q \triangleright \Delta'$, with Δ' balanced.

Why is our mechanisation of r-pass wrong?

$$\begin{array}{lrcl} \text{Type} & \alpha, \beta & ::= & ![S]; \alpha \mid ?[S]; \alpha \mid \mathbf{end} \mid \bot \\ \text{Typing} & \Delta & ::= & \cdot \mid \Delta, k : \alpha \end{array}$$

Subject Reduction: if $\Gamma \vdash P \triangleright \Delta$ with Δ balanced, and $P \rightarrow Q$, then there exists Δ' s.t. $\Gamma \vdash Q \triangleright \Delta'$, with Δ' balanced.

Rule $\texttt{throw}\ k\,[k'];P \mid \texttt{catch}\ k\,().Q \rightarrow P \mid Q^{k'}$ breaks subject reduction.

The Problem with Equating α-equivalent Terms

The idea behind R-PASS:

$$\begin{array}{l} \texttt{throw}\ k\,[k_0]; P \mid \texttt{catch}\ k\,(k_1).Q \rightarrow P \mid Q' \\ \quad \text{if} \quad \texttt{catch}\ k\,(k_1).Q \equiv_\alpha \texttt{catch}\ k\,(k_0).Q' \end{array}$$

The Problem with Equating α-equivalent Terms

The idea behind R-PASS:

$$\begin{array}{l} \texttt{throw}\ k\,[k_0]; P \mid \texttt{catch}\ k\,(k_1).Q \to P \mid Q' \\ \text{if} \quad \texttt{catch}\ k\,(k_1).Q \equiv_\alpha \texttt{catch}\ k\,(k_0).Q' \end{array}$$

k_0 cannot be free in Q

The Problem with Equating α-equivalent Terms

The idea behind R-PASS:

$$\begin{array}{l} \texttt{throw}\ k\,[k_0]; P \mid \texttt{catch}\ k\,(k_1).Q \rightarrow P \mid Q' \\ \text{if} \quad \texttt{catch}\ k\,(k_1).Q \equiv_\alpha \texttt{catch}\ k\,(k_0).Q' \end{array}$$

k_0 cannot be free in Q

We used a **standard** representation of binders, to mechanise a **non-standard, but correct** use of binders.

Mechanising the Revised System

Yoshida, N. and Vasconcelos, V. T. (2007). Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. Electronic Notes in Theoretical Computer Science, 171(4):73 – 93. Proceedings of the First International Workshop on Security and Rewriting Techniques (SecReT 2006)

Processes, Channels, and Polarities

A solution to the naive R-PASS is to distinguish **channel polarities** ([Gay and Hole, 2005])

$$p, q ::= + \mid -$$

$$\texttt{throw}\ k^{p}\,[k'^{q}]; P \mid \texttt{catch}\ k^{\overline{p}}\,().Q \to P \mid Q^{k'^{q}}$$

```
Inductive channel :=
| Ch of (kvar * polarity)
| Var of cvar.

Inductive proc : Set :=
| send : channel → exp → proc → proc
| receive : channel → proc → proc

| throw : channel → channel → proc → proc
| catch : channel → proc → proc

| nu_ch : proc → proc
...
```

Atoms: Separating Namespaces

```
Module CA := AtomScope Atom.Atom. (* Module of the atoms for channels *)
Module KA := AtomScope Atom.Atom. (* Module of the atoms for channel name *)
Module EA := AtomScope Atom.Atom. (* Module of the atoms for expressions *)

Notation cvar := (CA.var).
Notation kvar := (KA.var).
Notation evar := (EA.var).
```

Typing

Type $\alpha, \beta \quad ::= \quad ![S];\alpha \mid ?[S];\alpha \mid \textbf{end} \mid \ldots$

Typing $\quad \Delta \quad ::= \quad \cdot \mid \Delta, k : \alpha$

Judgement $\quad \Gamma \vdash P : \Delta$

```
Inductive tp : Set :=
  | input : sort → tp → tp
  | output : sort → tp → tp
  | ended : tp
  | ...
```

```
Inductive oft : sort_env → proc → tp_env → Prop :=
| t_send : forall G kt e P D S T,
    oft_exp G e S →
    oft G P (add kt T D) →
    oft G (send (chan_of_entry kt) e P) (add kt (output S T) D)
| ...
```

Linear Environments

$$\Delta ::= \cdot \mid \Delta, k : \alpha$$

```
Inductive env := Undef | Def of {finMap K → V}.

Definition add x t E :=
  if x \in dom E then Undef else upd x t E.
```

We defined operations on environments that contain **linear** channels.

Adding a channel that is already in the environment results in an **undefined** environment.

We use this fact pervasively in our mechanised proofs.

Subject Reduction

```
Theorem SubjectReductionStep G P Q D:
  oft G P D → P ⟶ Q → exists D', D ⇝ D' ∧ oft G Q D'.
```

```
Theorem SubjectReduction G P Q D:
  oft G P D → P ⟶* Q → exists D', oft G Q D'.
```

Mechanising a Proof of Subject Reduction

Using separate namespaces requires us to prove distinct **substitution lemmas** for every different kind of binder (expression, channel, shared channel).

Separate namespaces helps us avoid errors (e.g. using a channel instead of an expression), and simplifies proofs.

Linear environments allow us to make simplifying assumptions about <u>defined</u> environments.

Summary of Act I

- Deep Embedding binders allows us to fully control the calculus.
- LN requires a number of theorems and lemmas to prove our basic safety properties.
- EMTST (our tool) helps with nominal sets and environments.
- Next, we will explore what do we gain if we give up control (using shallow embeddings).

- Gay, S. and Hole, M. (2005). Subtyping for Session Types in the Pi Calculus. Acta Informatica, 42(2):191–225.

- Honda, K., Vasconcelos, V. T., and Kubo, M. (1998). Language primitives and type discipline for structured communication-based programming. In Hankin, C., editor, Programming Languages and Systems, pages 122–138, Berlin, Heidelberg. Springer Berlin Heidelberg.

- Yoshida, N. and Vasconcelos, V. T. (2007). Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. Electronic Notes in Theoretical Computer Science, 171(4):73 – 93. Proceedings of the First International Workshop on Security and Rewriting Techniques (SecReT 2006).